Scalable Mechanisms for Requirements Interaction Management

Martin S. Feather, Steven L. Cornford, Mark Gibbel
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109, USA
Martin.S.Feather@Jpl.Nasa.Gov
Steven.L.Cornford@Jpl.Nasa.Gov
Mark.Gibbel@Jpl.Nasa.Gov

Abstract

Capturing requirements, and managing tradeoffs among them, are critical yet complex activities. Well-designed computerized tools can effectively support these activities. A key challenge in construction of these support tools is how to scale them to handle a large volume of information. Particularly crucial are the ways in which large numbers of requirements and their interrelationships are presented to users. They need to be able to zoom in and out through the space of information so as to be able to see the big picture, and to locate and focus on specific details when needed. This paper describes a harmonious combination of techniques that support such scalability.

The techniques have been embodied in a NASA tool, DDP, for defect detection and prevention. They have been exercised in uses of this tool for requirements/risk tradeoffs, and population of this tool to capture institutional knowledge-bases of information.

1. Introduction

1.1. Requirements Interaction Management

Arriving at a set of requirements that is both beneficial (meets customer needs) and cost effective (can be implemented at reasonable cost) is an important early step in software development. [Karlsson & Ryan, 1997] reports two case studies of commercial projects that reveal the value of this. In the first study they found that by judiciously selecting a subset of requirements, 94% of the software system's maximum possible value to its customers could be met at 78% of the cost for implementing all requirements. The second study had figures of 95% of possible value at 75% of cost.

Determining the set of requirements to implement can be far from easy, especially when requirements interact with one another, or are still evolving. [Robinson et al, 1999] coin the term "Requirements Interaction Management" to cover this challenging area. Their definition reads:

> "Requirements Interaction Management is the set of activities directed towards the *discovery, management, and disposition of critical relationships among sets of requirements*."

Robinson et al survey seven projects that offer support for requirements interaction management, providing a wide variety of automated support for various aspects of the process.

The challenges stem from three sources: the sheer number of requirements and interactions, and the fact that requirements lie at the critical boundary between human understanding and machine representation, and the evolving nature of requirements as peoples' understanding improves. Human involvement is indispensable, yet manually applied processes quickly become tedious.

For example, Ryan & Karlsson employ the Analytic Hierarchy Process [Saaty, 1980] to establish prioritization of requirements by pairwise comparisons among them. In [Ryan & Karlsson 1977] they state, "The process was tedious, and was only realistic for small (<20) sets of requirements. Neither did it take account of the interdependencies between requirements that frequently occur in real projects." They go on to describe a prototype tool to support the process, now a commercial product Focal Point™ [Focal Point AB].

At NASA, we face similar challenges of requirements interaction management. One area of particular concern is risk management, where risks are the things that, should they occur, will lead to loss of requirements. Determining the impact of risks is done by quantitatively interrelating risks with requirements. A wide variety of

mitigation strategies are available to decrease risk, and thereby increase likelihood of attaining requirements. Determining the effectiveness of mitigations is done by quantitatively interrelating mitigations with risks. Appropriate selection of risk mitigations must balance their costs (budget, schedule, etc.) against their benefits (risk reduction). Like Ryan & Karlsson, we recognized early on that tool support was essential to support this process. In building this tool support we have found scaling to a large number of requirements, risks and mitigations is of key importance if the tool is to be effective. Particularly crucial are the ways in which large numbers of objects and their interrelationships are presented to users to assist their decision-making.

The purpose of this paper is to present realization of a harmonious combination of techniques that support such scalability. While these have been designed for our particular application (risk management for spacecraft flight systems), we feel that many of our approaches would have benefit to other tools that support requirements interaction management.

The remainder of this paper is organized as follows:

Section 2: An overview of NASA's Defect Detection and Prevention process, the objective of our tool support. This is the source of our examples throughout the paper.

Sections 3 – 6: key mechanisms that support scalability. Each is introduced in general terms, then the DDP solution is presented and discussed.

Section 7: Future Work

Section 8: Related Work and Conclusions

2. NASA's Defect Detection and Prevention (DDP) Process

NASA's Defect Detection and Prevention (DDP) process [Cornford, 1998] is a method for optimizing the collection of mitigation activities performed on a project. It allows one to perform overall risk management for flight systems. Since these prevention and detection activities incur costs (e.g., budget and schedule), their selection must tradeoff their costs against their benefits.

The principal elements of DDP are:

- Requirements – the desired goals.
- **F**ailure **M**odes (**FMs**) – the risks that, should they occur, cause loss of requirements.
- **P**reventions (typically design measures), **A**nalyses, process **C**ontrols (e.g., parts selection) and **T**ests (**PACTs**).
- Impact – a quantitative measure of how much loss of requirement is caused by an FM.
- Effectiveness – a quantitative measure of how much a PACT reduces the likelihood of a FM.

The DDP process requires tool support to be effective, because it must manage quantitative relationships amongst numerous elements. Furthermore, it must allow real-time brainstorming. Proof-of-concept studies employed a spreadsheet-based implementation. Their success established the value of the process, but revealed the cumbersome nature of the prototype spreadsheet-based implementation. NASA Code Q then funded an effort to develop a user-friendly tool for performing DDP. All of the examples in the sections that follow are drawn from this tool.

2.1. DDP: an example application.

A prototype of the tool was applied last year to several non-trivial design activity. A typical one of these culminated in the capture of 67 requirements, 105 FMs, and 93 PACTs. A representative screen display of this data is shown in Fig. 1. *(Note: the names of requirements, FMs and PACTs have been deliberately garbled so as to protect potentially sensitive information.).* This same data set is used as the source of all the examples shown in this paper.

The DDP tool employs familiar mechanisms to display various aspects of this information. Trees are used to display the hierarchy. For example, in Fig. 1 a portion of the requirements hierarchy can be seen in the window labeled "**Rqmts**" towards the top left of the screen. To its right, a similar window labeled "**FMs**" displays the hierarchy of FMs. Bar charts are used to display summary numerical information. For example, the expected degree to which the currently selected requirements will be met is shown in the upper bar chart labeled "**Requirements (log scale)**". Below it, a similar window labeled "**Risk Balance (log scale)**" displays the expected impact of each of the currently selected FMs. In the sections that follow we will also see two matrices, used to display interrelationships (the impact of FMs on Requirements, and the effectiveness of PACTs against FMs).

3. Views and Focus

3.1. Desiderata

Many systems offer multiple views into large information sets. Views can be used to present the same information in different ways, or to present different (or only marginally overlapping) subsets of information. For example, the widely popular Unified Modeling Language [UML] uses numerous different views into its requirements and design information. The Knowledge-Based Requirements Assistant (KBRA) [Czuchry & Harris, 1988] pioneered many of the multiple-view techniques for requirements engineering.

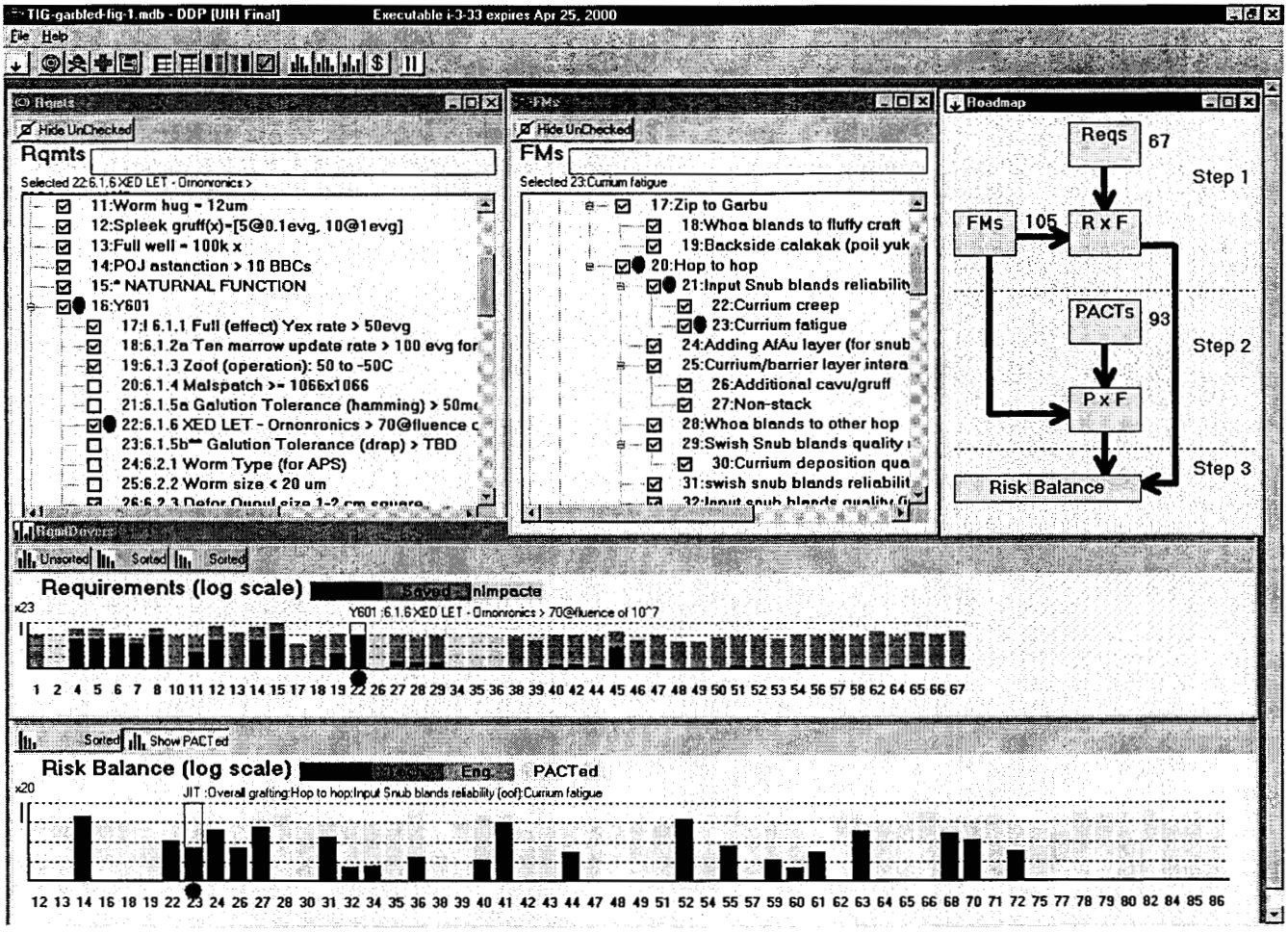


Figure 1. A typical DDP screen (names deliberately garbled)

It is important to be able to relate one view to another when switching between views. As the amount of information and the number of different views increases, the chance of confusion rapidly increases. For example, it is all too easy to loose track of the correspondence between the same elements presented in different views.

We have found the following practices ameliorate the problems of navigation between multiple views:

- Keeping the number of kinds of views to a minimum, reusing the same kind of view for multiple purposes.
- Automatically maintaining the equivalence between overlapping information displayed in different views.
- Where feasible, replicating another view's information in the current view.
- Where necessary, judiciously abbreviating information.
- Highlighting the current focus of concern.

3.2. DDP Realization

Keeping the number of kinds of views to a minimum, reusing the same kind of view for multiple purposes. In DDP there are four major kinds of views – tree views, matrix views, bar chart views, and item-specific views – through which to enter, inspect and adjust information. Tree views are used to display and restructure each of the three major classes of DDP hierarchically structured objects: requirements, FMs, and PACTs. Likewise, bar chart views are used to display the results of numerical computations on each of the current selections of these objects. Matrix views are used to enter, inspect and adjust the interrelationship information, specifically between requirements and FMs (how much of a requirement is lost if a FM occurs) and between PACTs and FMs (how effective a PACT will be at reducing the likelihood of a FM). Item-specific views display a single item at a time for data entry, inspection or adjustment. For example, when entering a new requirement, the item-specific view presents all the possible data fields

that can hold values associated with a requirement (title, importance, textual description, etc.).

Automatically maintaining the equivalence between overlapping information displayed in different views. Changes made to information through one view are immediately reflected in all the other views of that same information. For example, when the user adds a new requirement through the tree view, that same requirement is automatically added to the bar-chart view.

However, as the scale of the application increases, the large amount of information involved can make recalculations computationally expensive. For example, suppose the user adjusts the effectiveness of a PACT on a FM (the PACT's effectiveness is a numerical measure of the degree to which it reduces the likelihood of the FM). That FM might impact many requirements. The expected attainment of each of those requirements must be recomputed, so as to adjust their bars in the requirements' bar chart display accordingly. If this display is in "sorted" mode, it must be re-sorted.

We ameliorate this by running the automatic re-computation and redisplay in the background, so that the user can make additional changes without waiting for it to complete. Even this can be irksome when responsiveness of the display diminishes noticeably, and so we provide the user the option to turn off automatic re-computation and display, perform a batch of changes, and turn it back on again.

Where appropriate, replicating another view's information in the current view. Multiple views circumvent the impossibility of displaying all of a large amount of information in a single screen. However, if the views overly-fragment the information, relating one view to another can be hard. In response, we look for opportunities to replicate one view's information in another view. This is appropriate if it aids the user's navigation *and* can be achieved with little cost (notably, screen space).

For example, the portion of the tree view in Fig. 2 shows the names, hierarchy, checked / unchecked status, and expanded/unexpanded status of FMs.

Much of this is replicated (but in a different style) in the corresponding portions of the matrix views – for example, the Requirements x FMs matrix shown in Fig. 3. Requirements are listed in rows, FMs in columns. Grey-background cells display headers, names and aggregate values, while white-background cells display the numerical value of the relationship between the

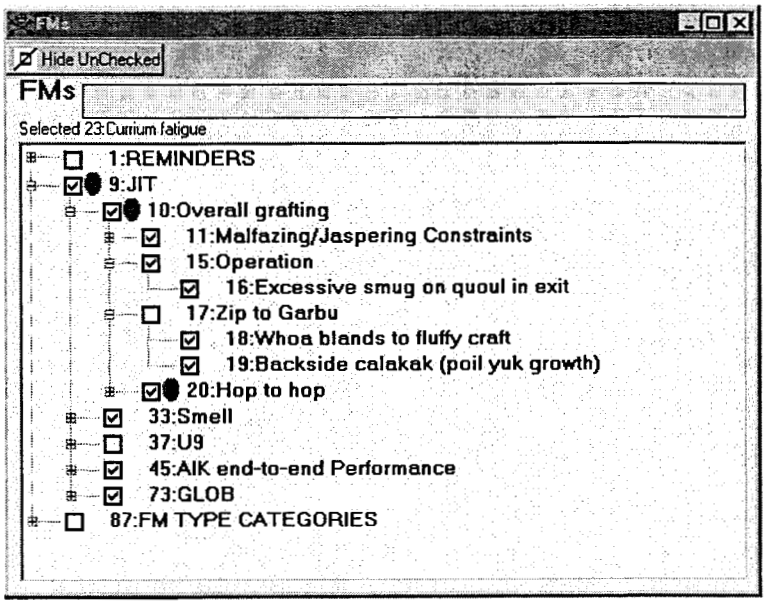
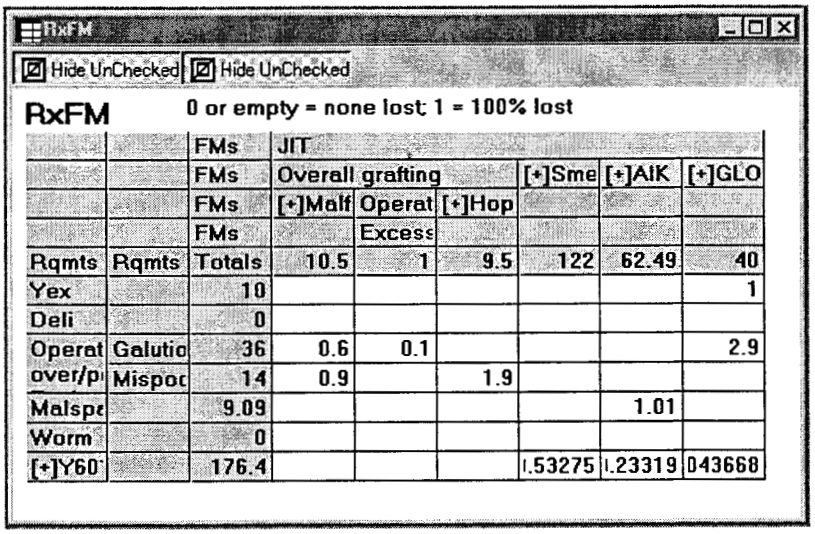


Figure 2. Tree view of FMs



		FMs	JIT					
		FMs	Overall grafting			[+]Sme	[+]AIK	[+]GLO
		FMs	[+]Malf	Operat	[+]Hop			
		FMs		Excess				
Rqmts	Rqmts	Totals	10.5	1	9.5	122	62.49	40
Yex		10						1
Deli		0						
Operat	Galutio	36	0.6	0.1				2.9
over/p	Mispoc	14	0.9		1.9			
Malspe		9.09					1.01	
Worm		0						
[+]Y60		176.4				1.53275	1.23319	043668

Figure 3. Matrix view of Requirements (rows) x FMs (columns)

corresponding FM and requirement (namely, the proportion of the requirement that would be lost if the FM were to occur).

Observe that the tree hierarchy is replicated by use of "header" cells in the matrix. For example the tree view shows **JIT** as a parent of **Overall grafting**, **Smell**, etc. The matrix view uses the topmost FM header cell to display **JIT** and spanning all the next-level header cells that are its included children, **Overall...**, **Smell**, etc.

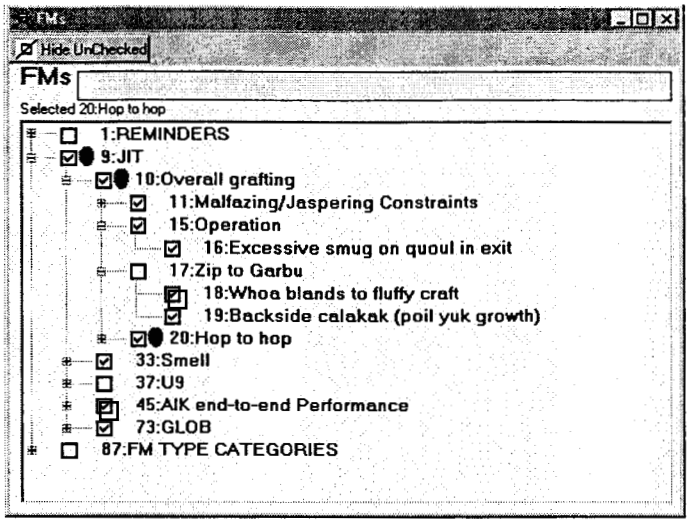


Figure 4. Hierarchy and focus highlighted in the tree view

The expanded/unexpanded statuses of tree nodes are also indicated in the matrix view, by prefixing unexpanded nodes' names with "**[+]**". Both of these add navigational value – within the matrix view the user can readily discern where a column falls within the hierarchy, and whether or not it corresponds to an unexpanded node. Also, note that both of these elements of information can be provided at a small cost of screen real estate.

Where necessary, judiciously abbreviating information. Abbreviating one view's information when showing it in another view is often a good approach. For example, while the tree view shows the full title of its nodes, the matrix view shows only as many characters as can squeeze into the cell size. Rows and/or columns can be manually resized to make additional room, but in general the matrix view simply lacks sufficient room to display full titles *and* a significant number of inner cells at the same time.

Dynamic techniques, such as popping up the full name when the mouse is positioned over a header cell, and offering a menu to get to detailed information by right clicking on a cell, are also employed. The user-interfaces of many existing tools make us familiar with these kinds of capabilities, and requirements interaction management tools can also take good advantage of them.

Highlighting the current focus of concern. DDP employs a notion of "focus" to draw users' attention to specific items in the various views. Its primary purpose is to aid in quickly focussing on some current item(s) of particular interest, but also serves to anchor users as they navigate from one view to another.

Fig. 4 shows the FM **Hop to Hop** as being the current focus, denoted by the blob just to the right of its check box. Note that its ancestry nodes (**Overall grafting**, and **JIT**) also have these same blobs. This highlighting of a node's ancestry is done automatically by DDP as soon as the user clicks on a leaf node.

Focus in the matrix view is shown in Fig. 5. The same FM that was in focus in Fig. 4 is in focus here - Hop to Hop. Its column of data cells is highlighted, as is its entire ancestry (header cells **JIT** and **Overall Grafting**). Similarly, a row of cells is highlighted corresponding to the requirement that is also in current focus.

A uniform coloring scheme pervades DDP – a different color for each of the categories Requirements, FMs and PACTs. The default scheme (not visible in this black-and-white reproduction) is:

- *Blue* for requirements
- *Red* for FMs
- *Green* for PACTs

Fig. 6 shows the corresponding focus on the "**Risk Balance**" bar chart (whose columns correspond to FMs). The blob beneath the third column corresponds to same FM as highlighted in the previous two figures. The

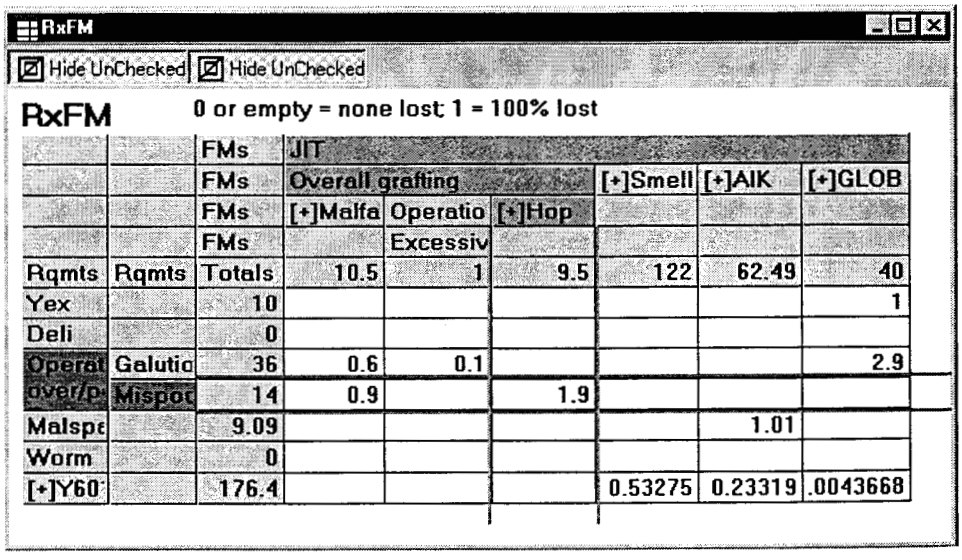


Figure 5. Hierarchy and focus highlighted in the matrix view

number beneath each FM bar is the same as that FM's number in the tree view. The presence of a "+" beneath a bar indicates it corresponds to an unexpanded parent FM.

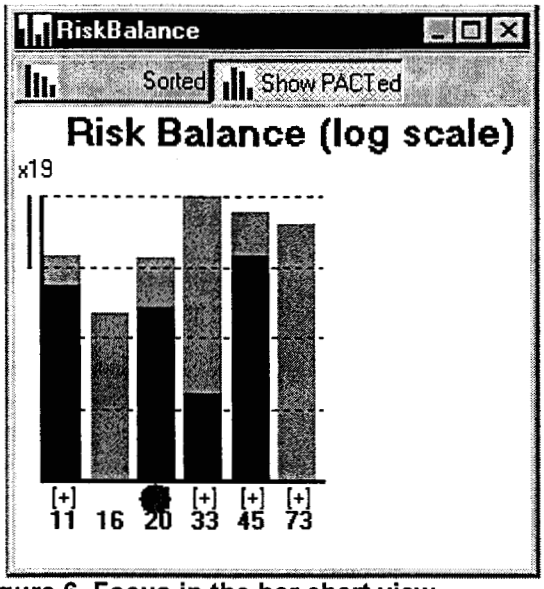


Figure 6. Focus in the bar chart view

4. Selection and Hierarchy

4.1. Desiderata

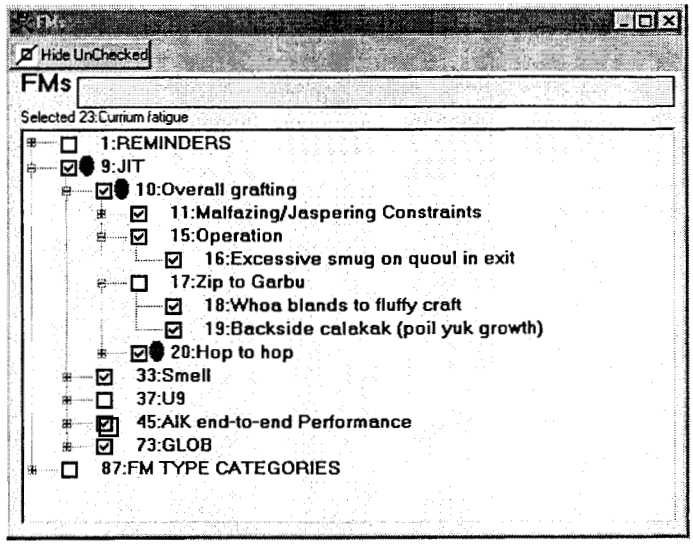
When handling voluminous amounts of information, it is important to provide users the ability to select subsets of information to work with. This permits them to focus their attention on the just the information relevant to the issues at hand, and avoid distraction of extraneous information. Without the capability to perform such selection, most obvious display regimes become unwieldy as the volume of information grows beyond trivial amounts. For example, users of the DDP precursor tool found they were continually scrolling across extensive matrices, because the full set of data was too large to fit onto a single screen.

4.2. DDP Realization

DDP provides the obvious capabilities for deleting information that is irrelevant to the task in hand. For example, the user can start from a pre-formulated set of FMs, and delete those that are not applicable to the current design. Those deleted FMs then play no further role in the user's world.

More interesting are the capabilities for *temporary* removal of information. Temporary implies the capability to reverse the removal. DDP uses "check boxes" to both control the removal process (information is temporarily removed by unchecking, and re-introduced by checking), and to identify what information is available for such re-introduction. DDP also employs the familiar notions of expanded and contracted trees, using Microsoft Windows™ conventions of "+" and "-" boxes for indicating and controlling the status of expansion of the displayed tree (a "+" box means there are undisplayed children of the corresponding node; clicking that box causes expansion to take place). Macintosh™ users are familiar with the same concept, but using arrow symbols in place of those boxes. Temporarily collapsing a sub-tree is a means to remove from view the details of its sub-structure. DDP harmoniously combines check boxes with tree structures, as described next.

Figure 2 (repeated for convenience below) illustrates DDP's combination of check boxes with tree structures.



Repeat of Figure 2. Tree view of FMs.

DDP provides selection and hierarchical structuring. They are combined so as to operate harmoniously together. A node of a tree is included in the current set of information if and only if its check box is checked, and all its ancestors' check boxes are also checked. For example, **Excessive smug on quoul in exit** is included, because its check box is checked, as is the check boxes of all its ancestors (namely, **Operation**, **Overall grafting**, and **JIT**). Conversely, **Whoa blands to fluffy craft** is not included, because although its check box is checked, not all of its ancestors' check boxes are checked (**Zip to Garbu** is unchecked). This allows for the rapid selection/de-selection of whole sub-trees at once, by simply checking/unchecking the root node.

Experience with using the tool showed that it is common to need to include a leaf node when one or more of its ancestors are unchecked. Given the above interpretation of checks combined with hierarchy, this implies that all the unchecked ancestor nodes must be checked so as to include the leaf node in the current selection. DDP accommodates this by automatically checking an unchecked node's ancestors whenever that node is checked. This is preferable to forcing the user to perform these steps manually, or disrupting the natural interpretation of checking and hierarchy.

In DDP, the effect of checking and tree expansion is carried over to other views, notably matrix and bar chart views. The tree view's checks and expansions determine the set of information and its depth of detail as displayed in those other views. In particular, the included nodes at the *fringe* of the tree are displayed. For example, Fig 3 shows the RxFM matrix corresponding to the above selection of FMs. It displays the six FMs that are the currently included nodes at the fringe of the FM tree, **Malfazing...**, **Excessive...**, **Hop to Hop**, **Smell**, **AIK...** and **GLOB**, corresponding to those same six selected leaf nodes in top-to-bottom order in the tree view. As discussed in the previous section, the hierarchy of the tree structure is also shown in the header cells of the matrix view, and the status of unexpanded parent nodes indicated by "**[+]**" prefixes on names. Note that in the matrix view the children of an unexpanded parent node are *not* displayed; to see these details, the user must return to the tree view.

the total impact of a FM is computed by summing over each requirement the requirement's weight times the FM's impact on that requirement, and multiplying the sum by the FM's likelihood. This yields the (a-priori) expected impact of that FM. The DDP tool (like its predecessor spreadsheet-based proof-of-concept) computes this information automatically. Furthermore, DDP offers visualizations of the information, as follows:

Requirements: a column is shown for each (included) requirement. A fragment of the requirements bar-chart display is shown below:

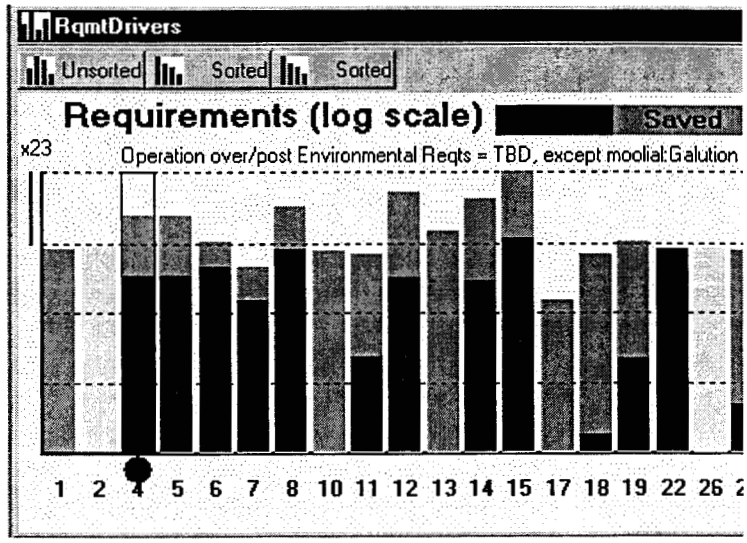


Figure 7. Bar-chart view of Requirements

5. Summarization and Aggregation

5.1. Desiderata

Summarization and aggregation are essential techniques to condense large amounts of detailed information.

5.2. DDP Realization

DDP manipulates several quantitative relationships on and between its data elements. Requirements have weights – a measure of their relative importance. FMs have a-priori likelihoods – their probability of occurrence if nothing is done to inhibit them. FMs are linked to requirements by "impact" – the proportion of the requirement that would be lost if the FM were to occur. PACTs are linked to FMs by "effect" – the proportional reduction in the FM's likelihood that application of the PACT would obtain.

The DDP process computes a variety of summarization information from this data. For example,

The height of a requirement's column indicates the logarithm of the requirement's weight. (DDP uses log scales because the application domain is concerned with reduction of risk to very small levels). Colors within each column are used to convey the current effect of the FMs and selected PACTs:

- Blue (shown here as light gray) indicates the portion of a requirement that is unimpacted by any FM.
- Green (shown here as dark gray) indicates the expected loss, due to FMs, that has been *prevented* by the current set of PACTs.
- Red (shown here as black) indicates the expected loss due to FMs, *despite* the current set of PACTs (since not all PACTs are perfect, some expectation of loss typically remains).

The left-to-right order of the columns is the same as the top-to-bottom fringe requirements in the tree view of requirements (and therefore the same as the left-to-right cells of the matrix view of requirements). Recall the use of the blob to highlight the column in current focus.

Like many tools, DDP provides for Pareto diagrams by offering the ability to sort the data displayed in bar charts. The figure below shows the left fragment of the same set of requirements sorted by the height of the red column, i.e., the amount of expected loss of requirement weight:

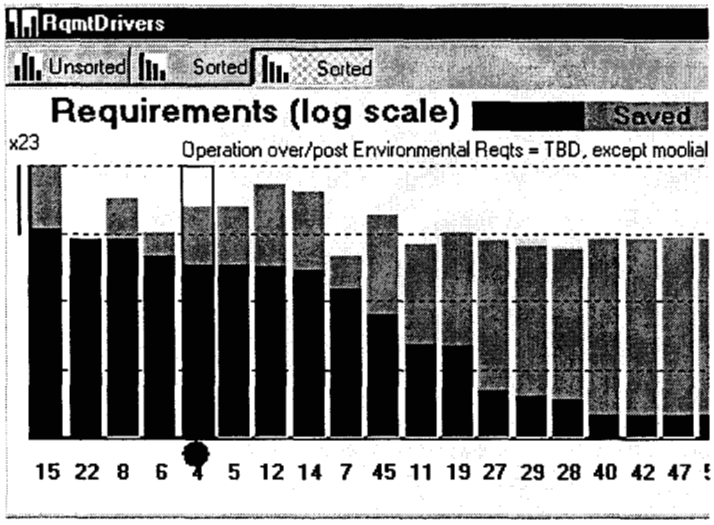


Figure 8. Sorted bar-chart view of Requirements.

Note that the highlighting of the in-focus requirement (via the blob beneath its column) shows where that requirement has moved to after sorting.

FMs: a column is shown for each (included) FM. A fragment of the FM bar-chart display is shown below:

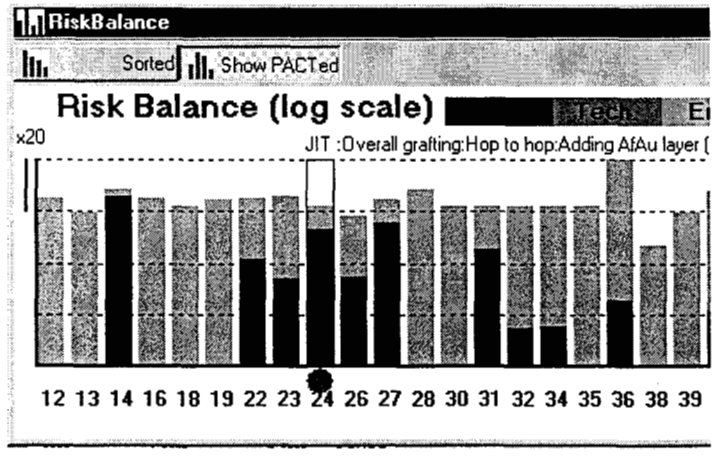


Figure 9. Bar-chart view of FMs

The height of a FM's column indicates the logarithm of the FM's summed impact on all included requirements. Colors within each column are used to convey the current effect of selected PACTs:

- Green (shown here as dark gray) indicates the expected loss due to that FM that has been *prevented* by the current set of PACTs.
- Red (shown here as black) indicates the expected loss due to FMs, *despite* the current set of PACTs.

As with the requirements bar chart, the left-to-right order is by default the same as the top-to-bottom order in the tree view, and an option to sort by height of red bars is available.

DDP also computes numerical summarizations of "aggregated" information. Section 4 discussed the tree-view's control of hierarchical structuring, and how that is replicated in the matrix view – the rows and columns of a matrix are the current "fringe" nodes of the corresponding trees. When a sub-tree contracted, the corresponding single row/column corresponds to that entire sub-tree. In particular, the numerical values in each cell in that row/column are *aggregations* of the values within the contracted sub-tree.

In the Requirement x FM matrix view, impact is expressed as the proportion of the requirement that would be lost. The aggregate requirement is composed of all its sub-tree's leaf requirements, and hence the aggregate proportional loss is a combination (suitably weighted by the relative importance of the requirements) of its leaf-node requirements' loss (it should be clear why we need mechanization to compute this!). For example, the upper portion of Fig. 10 shows requirement **Oper...** as unexpanded; its numerical values are aggregations of its constituent requirements. The lower portion shows this same node as expanded. The numerical impact values of its children are now shown.

The advantage of this is that users can work with collapsed sub-trees to get the "big picture", and zoom in to the details only when necessary. For example, the requirements bar chart view of the above information, with sub-tree collapsed, shows the height of the bar corresponding to the total failure mode impact on the weighted requirement (Fig 11). We see here that the red portion of this bar is relatively high as compared to most of the other visible bars. In this case the user probably *would* decide to zoom in to the details by expanding that sub-tree of requirements. If, however, the red bar had been relatively low, the user might have been content, and never needed to expand the sub-tree.

In summary, we perceive summarization and aggregation helps users to grasp the big picture, and to draw to their attention those areas where they need to investigate the details.

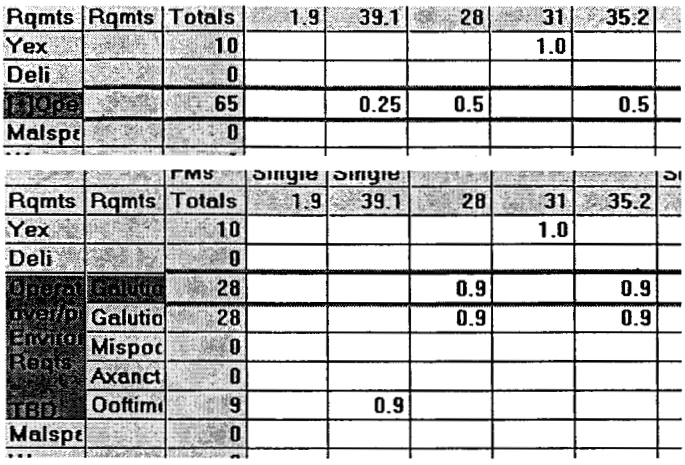


Figure 10. Collapsed (top) and expanded (bottom) sub-tree in matrix

6. Versions

6.1. Desiderata

Some form of version control is indispensable if requirements interaction management is to be scalable. Multiple versions can arise as a result of specialization (e.g., a baseline set of information is specialized for the task in hand), concurrency (e.g., when subdividing a large task among multiple people, who work in parallel), and investigation of alternatives (e.g., "what-if" explorations). Version control must provide an appropriate mix of isolation and dependency among different versions. Isolation is needed to ensure that modifications made in one version do not inadvertently propagate to others. Conversely, dependency is needed to ensure that modifications made in one version *do* propagate to other versions that are in an appropriate correspondence to the modified version.

6.2. DDP Realization

We foresee all these issues arising in the application of DDP to risk tradeoff analysis. For example, a particular project will want to inherit and specialize pre-built trees of typical FM. Multiple people will want to be able work on different parts of the requirements elicitation, and thereafter merge their results. Users will want to study different choices of sets of PACTs side-by-side.

To date, DDP's support for these aspects of version control is only partially complete. DDP data is already set up to be cumulative, i.e., any changes (whether they be additions, updates or deletions) are recorded as additional data, associated with a version. For example, if a FM is deleted, then internally the data record corresponding to that FM is retained, and an *additional* data item is created, indicating the deletion and the version ID of the version of information in which that deletion took place.

7. Future Work

DDP's design continues to evolve. We anticipate the need to work on the following areas, all of which will likely further exercise the need for scalability:

Yet more data: As more institutional data is gathered, the data sets will grow further. Additional techniques may be needed if scaling is to accommodate an order of magnitude or more increase in information.

Process: At present the DDP tool offers only rudimentary guidance on how to proceed through the major steps of the DDP process. This could be improved, particularly to help users navigate within, and between, stages of brainstorming, decision-making, etc.

Merging: We know we need to add support for concurrent DDP activities, followed by a merge process to combine the separate results. Dr. Barbara Gannod of Arizona State University has studied the issue with DDP in mind, and her development of tool support for this is in progress.

Enrichment of the types of data and relationships. We may choose to augment FMs with probabilistic distributions, and extend the computations accordingly. We may choose to cross-reference requirements to the design structure. Enrichments such as these may well induce the need for further kinds of views. Once we include cost information for PACTs, we can begin to automate the search for optimal sets of PACTs.

Integrate with design processes – DDP's prime focus is on planning for risk management. DDP could benefit from integration with design activities, and the processes and support tools that exist for them. Note that the outcome of DDP is a set of choices of PACTs (risk mitigants). In the course of performing those PACTs we

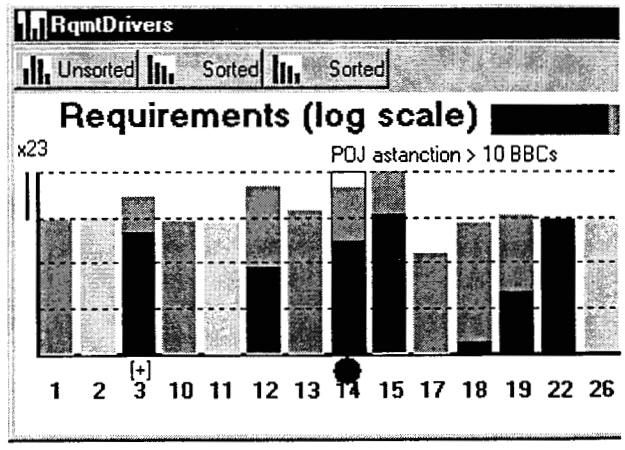


Figure 11. Collapsed sub-tree in bar chart

might learn additional information that should be fed back into a DPP-like process – i.e., DDP might become an ongoing activity continually kept in step with development.

8. Related Work and Conclusions

In the arena of requirements interaction management we are aware of several other efforts beginning to address the issues of scalability.

All processes for requirements interaction management share the concern of how to deal with activities that incur a cost that grows more-than-linearly with the number of requirements. For example, requirements prioritization or conflict resolution is often based on considering all pairwise combinations of requirements, an $O(n^2)$ operation for n requirements.

[Robinson & Pawlowski, 1998] present a method that circumvents this quadratic growth by identifying "root" requirements ("which represent key concepts from which other requirements are derived through elaboration"). The end result of their method is a classification hierarchy of requirements. Their method helps construct this hierarchy. [Ryan & Karlsson, 1997] also consider strategies to address this same problem, e.g., taking advantage of the knowledge of mutual exclusion between requirements to automatically rule out all subsets in which both those requirements are simultaneously included. In the WinWin system [Boehm & In, 1996], this same issue is addressed by use of a quality attribute hierarchy to narrow attention to those requirements that conflict over the same quality attribute.

DDP would appear to face these same problems in completing the requirements x FM, and PACT x FM matrices. However, DDP users seem adept at organizing their information into hierarchies, such that the subsets of information that could plausibly interact are easy to narrow down. Also, the tool's use of summarization and aggregation (section 5) can help users recognize when they need not descent to lower levels of detail.

Graphical visualizations of requirements information are also emerging. [Park et al, 1999] present their Distributed Collaboration Priorities Tool (DCPT), a companion tool to WinWin that helps prioritization of development items and risk reduction. DCPT makes repeated use of a variety of styles of 2-D charts whose axes represent orthogonal continuous-valued measures (e.g., importance vs. difficulty; probability vs. loss). These are used to display information to users, to help them make selections (e.g., based on Return-On-Investment as seen in the Importance vs. Difficulty chart). Focal Point™ [Focal Point AB], likewise uses 2-D charts to show importance (value) against cost, and integrates check-boxes into this same chart for intuitive use and ease of (de)selection.

In DDP the primary objective is to choose a cost-effective set of PACTs whose net effect will reduce risk to an acceptable level. However, to date cost information (schedule, budget, personnel or whatever) is *not* recorded! This leaves it to the insight of users to make the cost/benefit tradeoffs. The main value they obtain from the tool is the understanding of the expected loss of requirements. Perhaps because of this state of affairs, we have the found bar chart displays described in Section 5 to be the most appropriate means to portray quantitative aspects.

DCPT use the aforementioned 2-D presentations for assisting multiple users to reconcile differences (e.g., over the spread of opinions on the importance vs. difficulty of a single requirement).

To date we have used DDP primarily in a *non*-distributed fashion, were the multiple stakeholders are together at the same time and place. This seems to circumvent the need to provide elaborate support for maintaining multiple conflicting opinions simultaneously. Instead, we find that users almost always are solve the problem in one of the following ways:

- Resolve the issue by immediate discussion, yielding a single agreed upon result, which can then be entered.
- Defer decision until later. The tool offers the option of entering a non-numerical value into matrices (e.g., "**??? this is either 0.1 or 0.5**"), which the tool ignores for performing numerical calculations, but displays for users to locate and respond to at a later date.
- Realize that they are talking about related, but non-identical, cases. Usually this results in a bifurcation (e.g., what began as a disagreement on the extend to which a particular FM impacts a particular requirement leads to refining that FM into two children of the original, each with their own impact value). DDP's tree manipulations support this.

Graph structures, rather than simply tree structures, are featured in several groups' methodologies for requirements interaction management. For example, the KAOS methodology employs graphs to represent a semantic network of goals, constraints and objects. The KAOS support environment - GRAIL [Darimont et al, 1997] - provides a graphical editor for these structures. GRAIL uses multiple views – the graph view for high-level view of requirements, a text-based view for entry/editing of the details, and a hypertext view for navigating through a textual representation. As yet, DDP has not needed to display graph structures, so we do not have a direct equivalent of this kind of view.

We have discussed and illustrated scalability mechanisms, embodied in a working tool that supports

NASA's Defect Detection and Prevention (DDP) process. Experience with using the tool on real examples had convinced us of the need for addressing scalability, and led to our development of these capabilities.

9. Acknowledgements

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space administration through Code Q. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

10. References

[Boehm & In, 1996] B. Boehm & H. In. Identifying Quality-Requirement Conflicts. *IEEE Software*, March 1996, 25-36.

[Cornford, 1998] S. Cornford. Managing Risk as a Resource using the Defect Detection and Prevention process. *International Conference on Probabilistic Safety Assessment and Management*, September 13-18, 1998.

[Czuchry & Harris, 1988] A.J. Czuchry, Jr. & D.R. Harris. KBRA: A new paradigm for requirements engineering. *IEEE Expert*, Winter 1988. 3(4):21-35.

[Darimont et al, 1997] R. Darimont, E. Delor, P. Massonet & A. van Lamsweerde. GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering. International Conference on Software Engineering, May 1997, 612-613.

[Focal Point AB] Focal Point™, a trademark of Focal Point AB http://www.focalpoint.se

[Karlsson & Ryan, 1997] J. Karlsson & K. Ryan. A Cost-Value Approach for Prioritizing Requirements. *IEEE Software*, Sept./Oct. 1997, 67-74.

[Park et al, 1999] J-W. Park, D. Port, B. Boehm, & H. In. Supporting Distributed Collaborative Prioritization for WinWin Requirements Capture and Negotiations. *International Workshop on Process support for Distributed Team-Based Software Development (PDTSD'99) in World MultiConference SCI/ISAS '99*, Orlando, August, 1999

[Robinson et al., 1999] W.N. Robinson, S.D. Pawlowski & S. Volkov. Requirements Interaction Management. Working Paper of Georgia State University – contact the lead author at:. wrobinson@gsu.edu

[Robinson & Pawlowski, 1998] W.N. Robinson, S. Pawlowski. Surfacing Root Requirements Interactions from Inquiry Cycle Requirements. *International Conference on Requirements Engineering*, IEEE, April 6-10, 1998, Colorado Springs, CO, pp. 82-89

[Ryan & Karlsson, 1997] K. Ryan & J. Karlsson. Prioritizing Software Requirements in an Industrial Setting. *International Conference on Software Engineering*, May 1997, IEEE Computer Society, 564-565.

[Saaty, 1980] T.L. Saaty. *The Analytic Hierarchy Process*. McGraw-Hill, New York, 1980.

[UML] http://www.rational.com/uml